IS5 in R: The Standard Deviation as a Ruler and the Normal Model (Chapter 5)

Nicholas Horton (nhorton@amherst.edu)

2025-01-23

Table of contents

Introduction and background

This document is intended to help describe how to undertake analyses introduced as examples in the Fifth Edition of *Intro Stats* (2018) by De Veaux, Velleman, and Bock. This file as well as the associated Quarto reproducible analysis source file used to create it can be found at http://nhorton.people.amherst.edu/is5.

This work leverages initiatives undertaken by Project MOSAIC (http://www.mosaic-web.org), an NSF-funded effort to improve the teaching of statistics, calculus, science and computing in the undergraduate curriculum. In particular, we utilize the `mosaic` package, which was written to simplify the use of R for introductory statistics courses. A short summary of the R needed to teach introductory statistics can be found in the mosaic package vignettes (https://cran.r-project.org/web/packages/mosaic). A paper describing the mosaic approach was published in the *R Journal*: https://journal.r-project.org/archive/2017/RJ-2017-024.

We begin by loading packages that will be required for our analyses.

```r
library(mosaic)
library(tidyverse)
```

Chapter 5: The Standard Deviation as a Ruler and the Normal Model

```r
library(mosaic)
library(readr)
library(janitor)
WomenHeptathlon2016 <-
  read_csv("http://nhorton.people.amherst.edu/is5/data/Womens_Heptathlon_2016.csv") |>
  janitor::clean_names()
```

By default, `read_csv()` prints the variable names. These messages were suppressed using the `message: false` code chunk option to save space and improve readability. Here we use the `clean_names()` function from the `janitor` package to sanitize the names of the columns (which would otherwise contain special characters or white space).

```r
# page 123
df_stats(~ long_jump, data = WomenHeptathlon2016)
```

```
   response  min   Q1 median   Q3  max     mean        sd  n missing
1 long_jump 5.51 6.08   6.19 6.31 6.58 6.169655 0.2474655 29       2
```

```r
df_stats(~ x200m, data = WomenHeptathlon2016)
```

```
  response   min    Q1 median    Q3   max     mean        sd  n missing
1    x200m 23.26 24.12   24.6 24.99 26.32 24.58207 0.6544975 29       2
```

```r
with(WomenHeptathlon2016, stem(x200m))
```

```

  The decimal point is at the |

  23 | 3
  23 | 589
  24 | 011123334
  24 | 5667789
```

```
  25 | 00112444
  25 |
  26 | 3
```

```
# the `stem()` function doesn't have a `data = ` option
with(WomenHeptathlon2016, stem(long_jump))
```

```
  The decimal point is 1 digit(s) to the left of the |

  54 | 1
  56 | 2
  58 | 181
  60 | 0588002569
  62 | 023501145
  64 | 38158
```

Section 5.1: Using the Standard Deviation to Standardize Values

```
filter(WomenHeptathlon2016, last_name == "Thiam") |>
  tibble()
```

```
# A tibble: 1 x 9
  first_name last_name x200m long_jump x800m high_jump x100m_hurdles javelin
  <chr>      <chr>     <dbl>     <dbl> <dbl>     <dbl>         <dbl>   <dbl>
1 Nafissatou Thiam      25.1      6.58  137.      1.98          13.6    53.1
# i 1 more variable: shot_put <dbl>
```

```
# calculate z-score with mean and sd from df_stats
(6.58 - 6.17) / .247 # long jump
```

```
[1] 1.659919
```

```
filter(WomenHeptathlon2016, last_name == "Johnson-Thompson") |>
  tibble()
```

```
# A tibble: 1 x 9
  first_name last_name     x200m long_jump x800m high_jump x100m_hurdles javelin
  <chr>      <chr>         <dbl>     <dbl> <dbl>     <dbl>         <dbl>   <dbl>
1 Katarina   Johnson-Thom~  23.3      6.51  130.      1.98          13.5    36.4
# i 1 more variable: shot_put <dbl>
```

The `tibble()` function converts an object into a variant of a "data frame" (you may also see the use of `data.frame()` for this purpose.)

Note the difference when we pipe the results of `filter()` into the `data.frame()` function.

```r
filter(WomenHeptathlon2016, last_name == "Johnson-Thompson") |>
  data.frame()
```

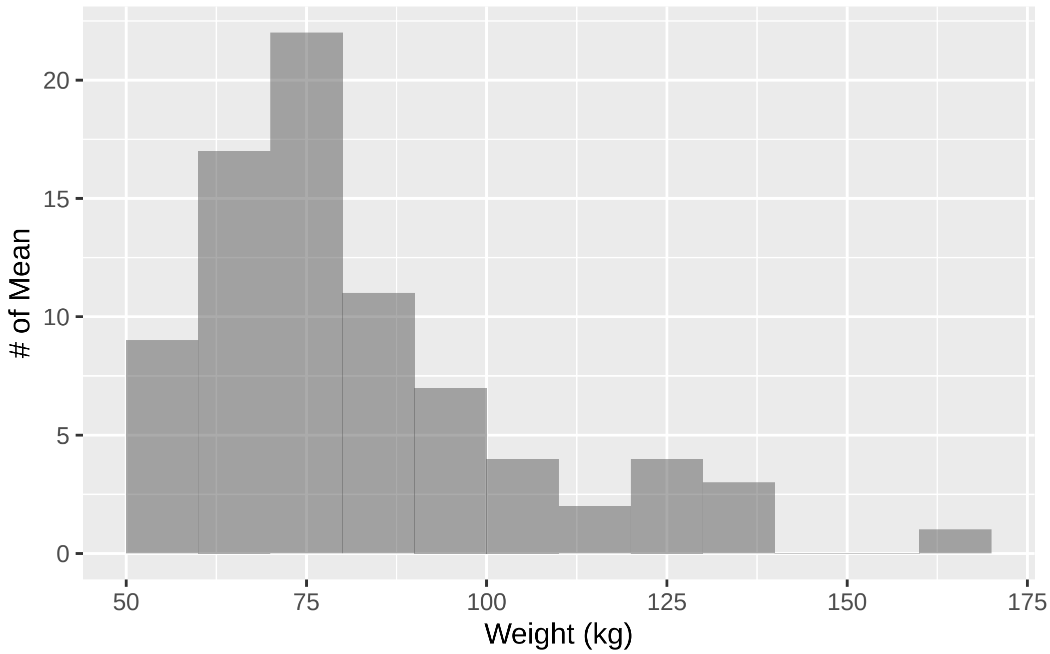
```
  first_name        last_name x200m long_jump  x800m high_jump x100m_hurdles
1   Katarina Johnson-Thompson 23.26      6.51 130.47      1.98         13.48
  javelin shot_put
1   36.36    11.68
```

Section 5.2: Shifting and Scaling

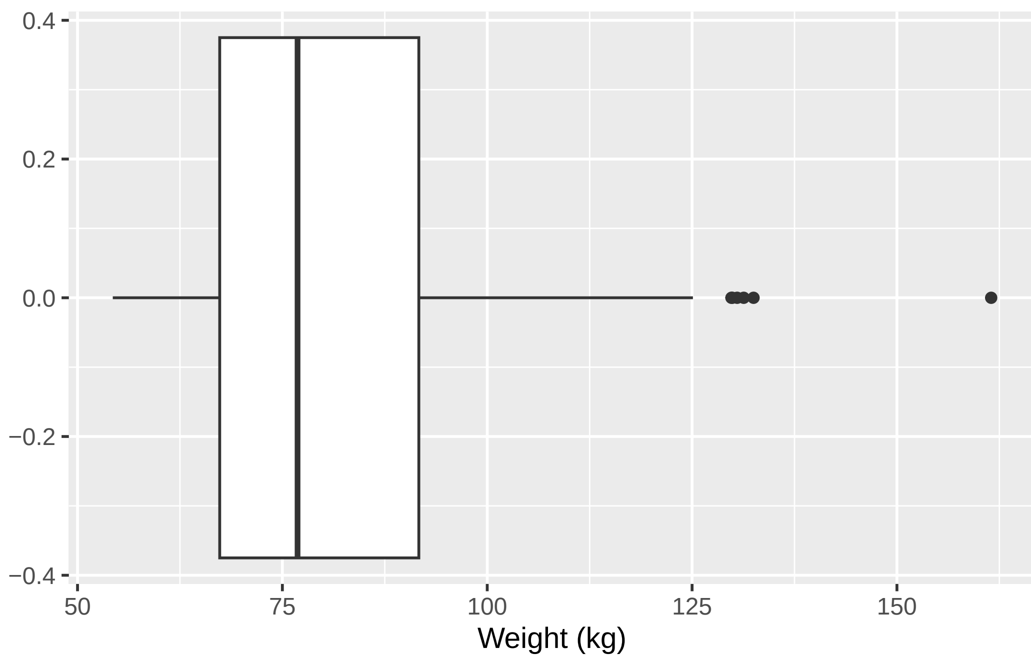
Shifting to Adjust the Center

We begin by reading in the data.

```r
MenWeight <- read_csv("http://nhorton.people.amherst.edu/is5/data/Mens_Weights.csv") |>
  janitor::clean_names()
# Figure 5.2, page 125
gf_histogram(~ weight_in_kg, data = MenWeight, binwidth = 10, center = 5) |>
  gf_labs(x = "Weight (kg)", y = "# of Mean")
```




```r
gf_boxplot(~ weight_in_kg, data = MenWeight, xlab = "Weight (kg)")
```

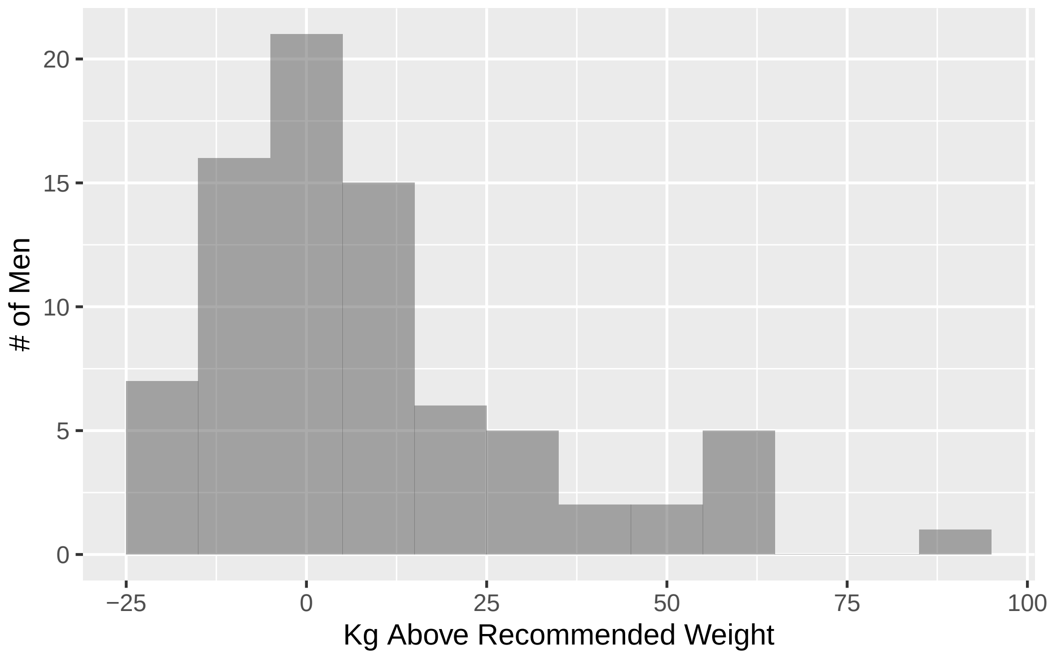


As noted previously, a single boxplot is not a good way to display the data (boxplots are better for comparisons).

```
df_stats(~ weight_in_kg, data = MenWeight)
```

```
      response  min    Q1 median    Q3   max     mean       sd  n missing
1 weight_in_kg 54.3 67.35  76.85 91.65 161.5 82.35625 22.26881 80       0
```

```
# Figure 5.3
gf_histogram(~ (weight_in_kg - 74), data = MenWeight, binwidth = 10) |>
  gf_labs(x = "Kg Above Recommended Weight", y = "# of Men")
```



Rescaling to Adjust the Scale

Let's review the data from the `MenWeight` dataset.

```
df_stats(~ weight_in_kg, data = MenWeight)
```

```
      response  min    Q1 median    Q3   max     mean       sd  n missing
1 weight_in_kg 54.3 67.35  76.85 91.65 161.5 82.35625 22.26881 80       0
```

```
df_stats(~ weight_in_pounds, data = MenWeight)
```

```
          response    min     Q1 median     Q3   max     mean       sd  n
1 weight_in_pounds 119.46 148.17 169.07 201.63 355.3 181.1838 48.99137 80
  missing
1       0
```

```r
MenWeight |>
  head() # There are two variables: weight_in_kg and weight_in_pounds.
```

```
# A tibble: 6 x 2
  weight_in_kg weight_in_pounds
         <dbl>            <dbl>
1        107.              236.
2         95.7             211.
3         68.9             152.
4         60.3             133.
5         60.4             133.
6         69.7             153.
```

```r
# Each observation has a value for each.
nrow(MenWeight)
```

```
[1] 80
```

```r
MenLonger <- MenWeight |>
  tidyr::pivot_longer(cols = starts_with("weight"),
               values_to = "weight",
               names_to = "weighttype")
MenLonger |>
  head() # The two variables are weighttype and weight.
```

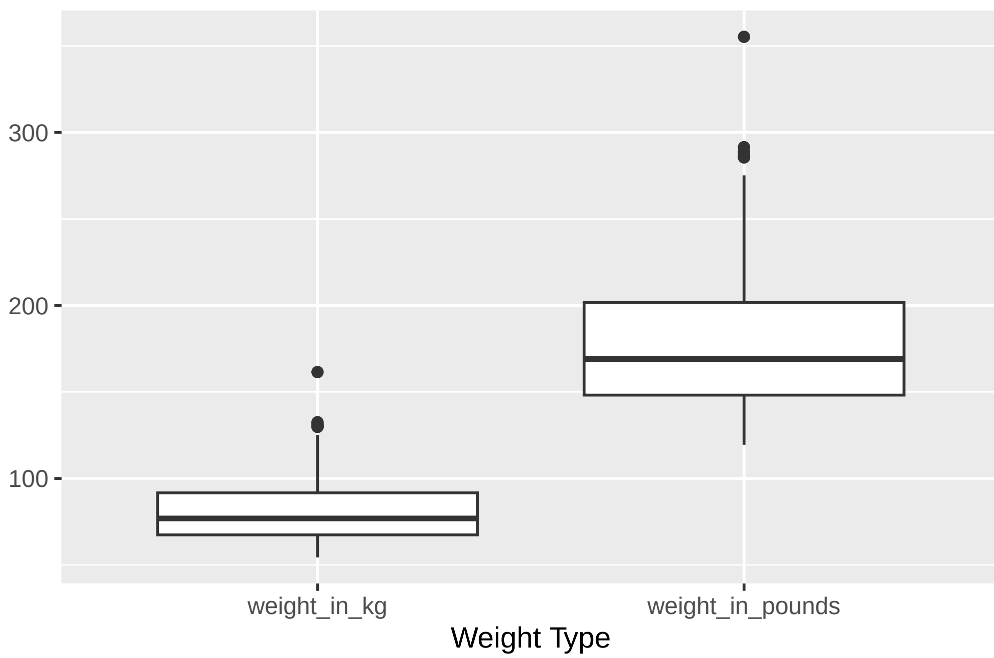
```
# A tibble: 6 x 2
  weighttype       weight
  <chr>             <dbl>
1 weight_in_kg      107.
2 weight_in_pounds  236.
3 weight_in_kg       95.7
4 weight_in_pounds  211.
5 weight_in_kg       68.9
6 weight_in_pounds  152.
```

```
# weighttype is a categorical variable that is either in kg or pounds
nrow(MenLonger) # Each observation from before is now two rows
```

```
[1] 160
```

Here we use the `tidyr::pivot_wider()` function to transform the dataset into the needed format, which can be seen with the `head()` function. This is an important but more complicated data wrangling idiom that we will use to reshape datasets.

```
MenLonger |>
  gf_boxplot(weight ~ weighttype) |>
  gf_labs(x = "Weight Type", y = "")
```



We see the use of `GOAL(Y ~ X)` as an example of the general modeling language for two variables in the `mosaic` package.

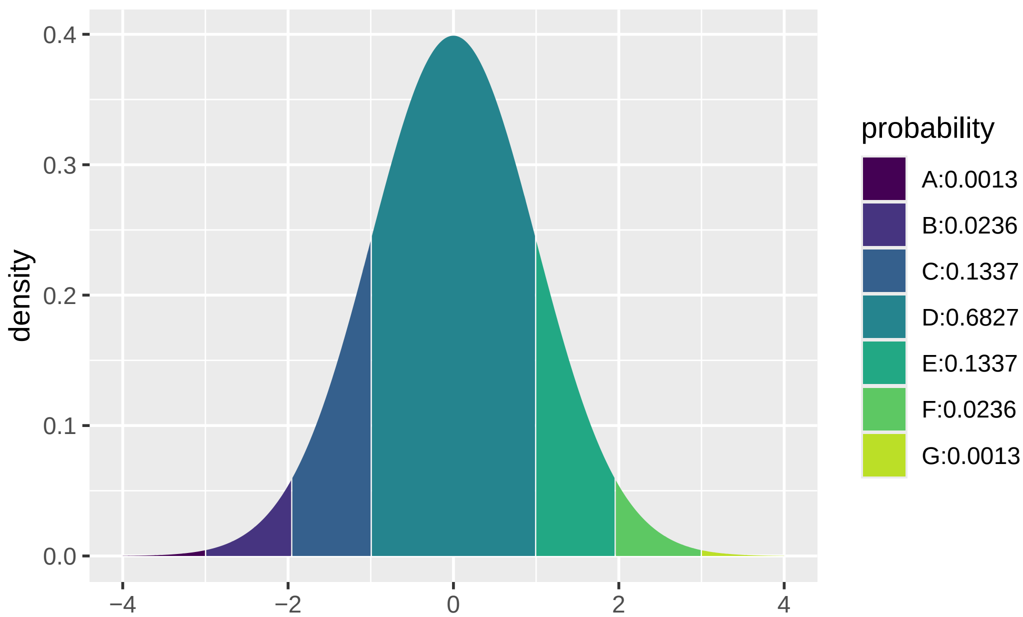
Shifting, Scaling, and the z-Scores

Section 5.3: Normal Models

The 68-95-99.7 Rule

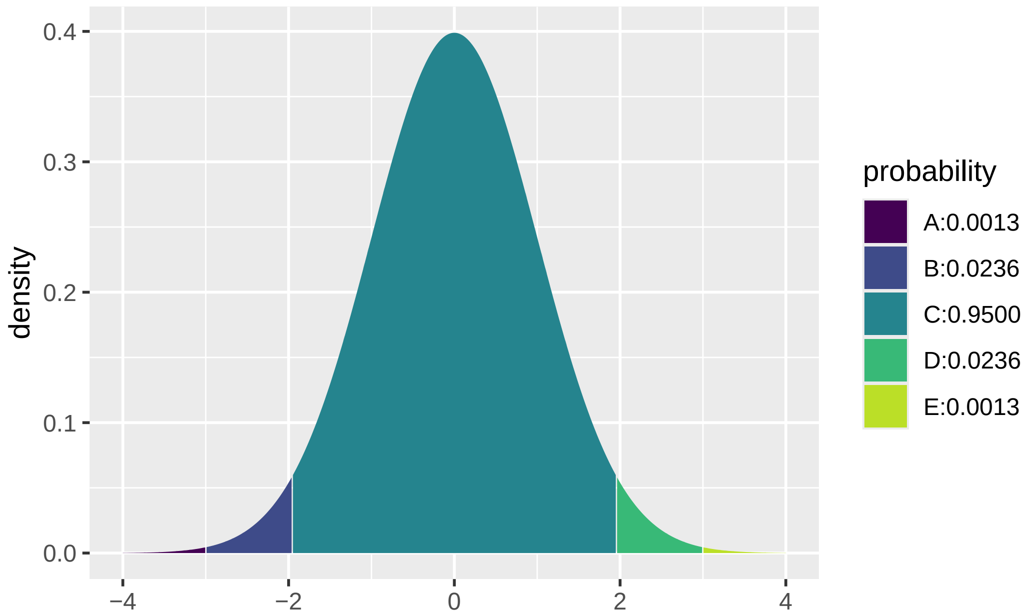
See display on page 129.

```
# Figure 5.6
# 1, 2 (1.96), and 3 SD's
xpnorm(c(-3, -1.96, -1, 1, 1.96, 3), mean = 0, sd = 1, verbose = FALSE)
```



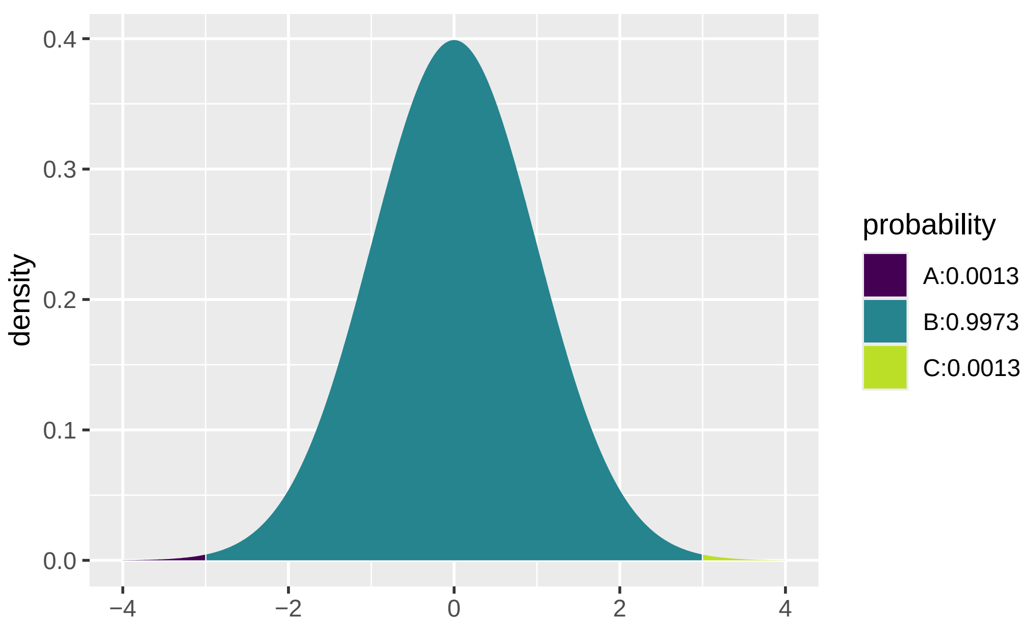

```
[1] 0.001349898 0.024997895 0.158655254 0.841344746 0.975002105 0.998650102
```

```
# 2 (1.96) and 3 SD's
xpnorm(c(-3, -1.96, 1.96, 3), mean = 0, sd = 1, verbose = FALSE)
```




```
[1] 0.001349898 0.024997895 0.975002105 0.998650102
```

```r
# 3 SD's
xpnorm(c(-3, 3), mean = 0, sd = 1, verbose = FALSE)
```

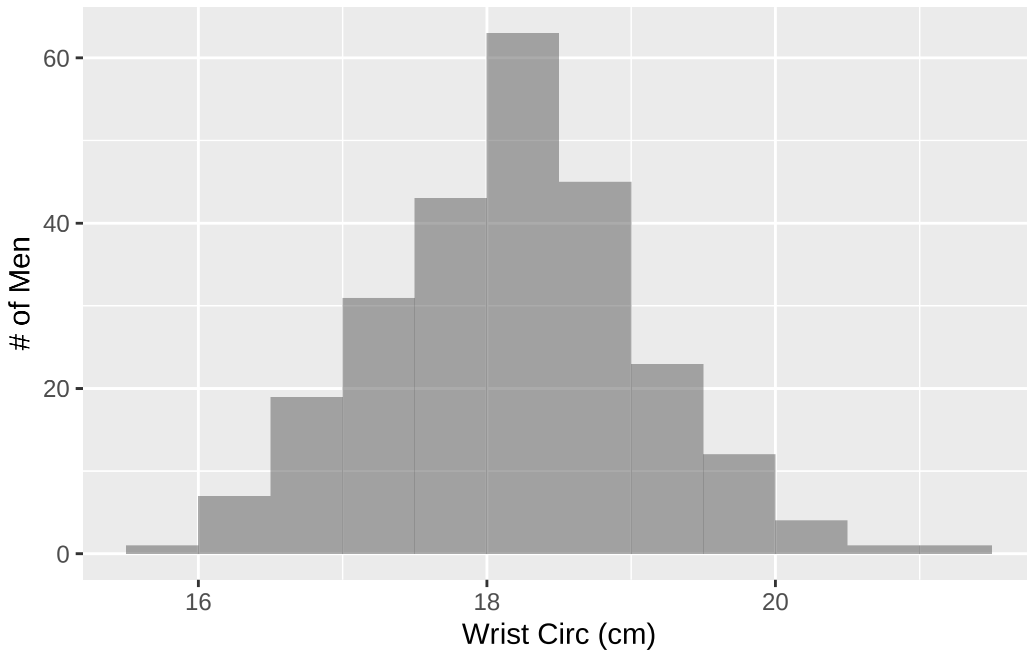



```
[1] 0.001349898 0.998650102
```

Example 5.4: Using the 68-95-99.7 Rule

We begin by reading in the data.

```
BodyFat <- read_csv("http://nhorton.people.amherst.edu/is5/data/Bodyfat.csv")
gf_histogram(
  ~ Wrist,
  data = BodyFat, binwidth = .5,
  center = -.25
) |>
  gf_labs(x = "Wrist Circ (cm)", y = "# of Men")
```

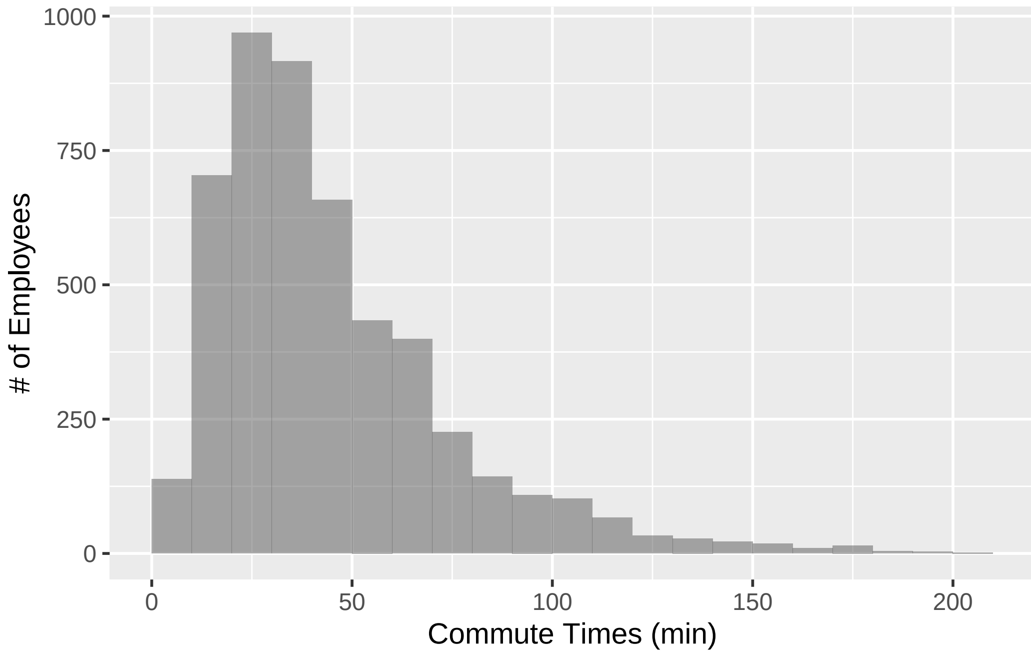


Random Matters

Starts on page 133.

```
Commute <-
  read_csv("http://nhorton.people.amherst.edu/is5/data/Population_Commute_Times.csv") |>
  janitor::clean_names()

gf_histogram(~ commute_time, data = Commute, binwidth = 10, center = 5) |>
  gf_labs(x = "Commute Times (min)", y = "# of Employees")
```




```
set.seed(2143) # To ensure we get the same values when we run it multiple times
num_sim <- 10000 # Number of simulations
samp_size <- 100 # Desired sample size


mean(~ commute_time, data = sample(Commute, size = samp_size)) # Mean of one random sample
```

```
[1] 45.79
```

```
mean(~ commute_time, data = sample(Commute, size = samp_size)) # Mean of another random sampl
```

```
[1] 44.7
```

The `mosaic::do()` command allows us to run a command multiple times, saving the result as a data frame.

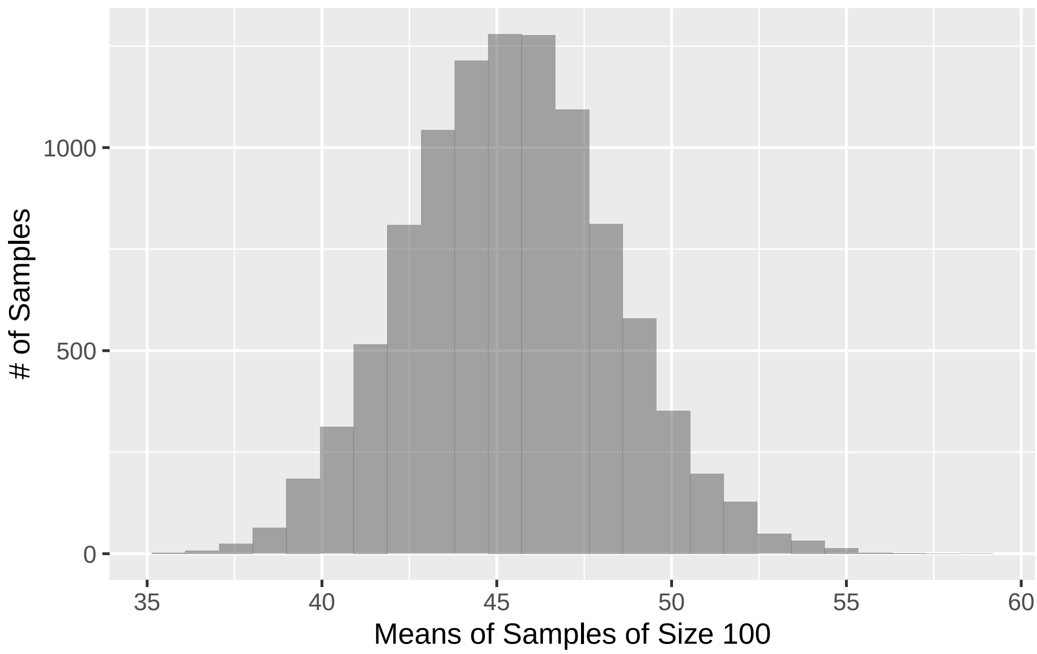
```
do(2) * mean(~ commute_time, data = sample(Commute, size = samp_size))
```

```
   mean
1 47.43
2 45.97
```

```
# For the visualization, we use do() 10,000 times
Commute_sample <- do(num_sim) * mean(~commute_time, data = sample(Commute, size = samp_size))
```

The `do()` function generates 10,000 samples of size `samp_size` and for each calculates the sample mean.

```
gf_histogram(~ mean, data = Commute_sample) |>
  gf_labs(x = "Means of Samples of Size 100", y = "# of Samples")
```



Section 5.4: Working with Normal Percentiles

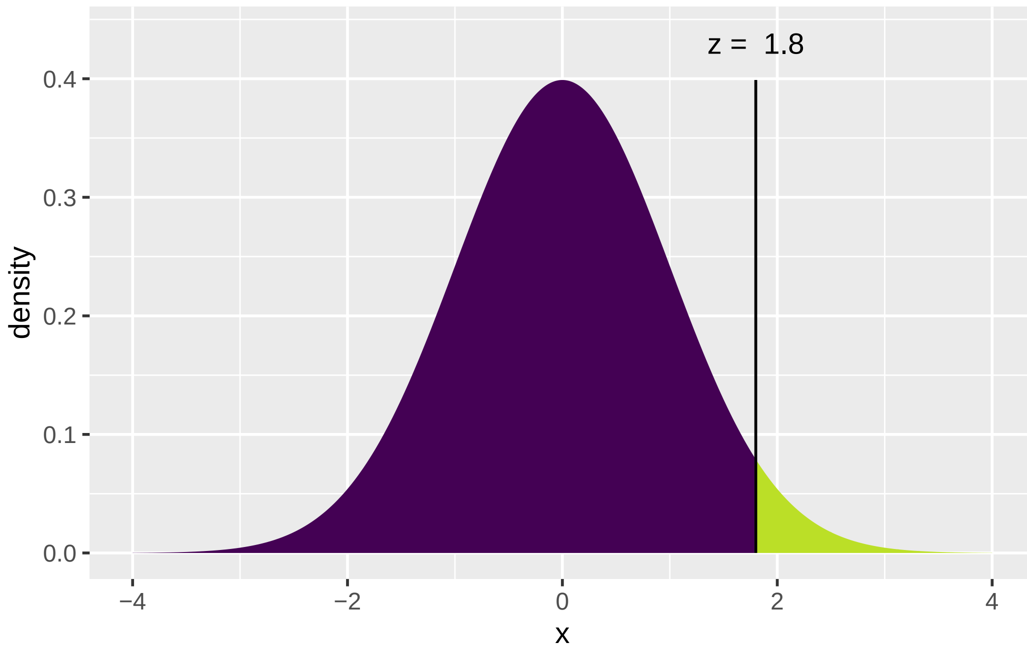
The `pnorm()` function calculates normal probabilities. The `xpnorm()` function from the mosaic package adds a graphical depiction and additional output that may be helpful to new users.

```
xpnorm(1.8, mean = 0, sd = 1)
```

```
If X ~ N(0, 1), then

    P(X <= 1.8) = P(Z <= 1.8) = 0.9641
```

```
    P(X >  1.8) = P(Z >  1.8) = 0.03593
```




```
[1] 0.9640697
```

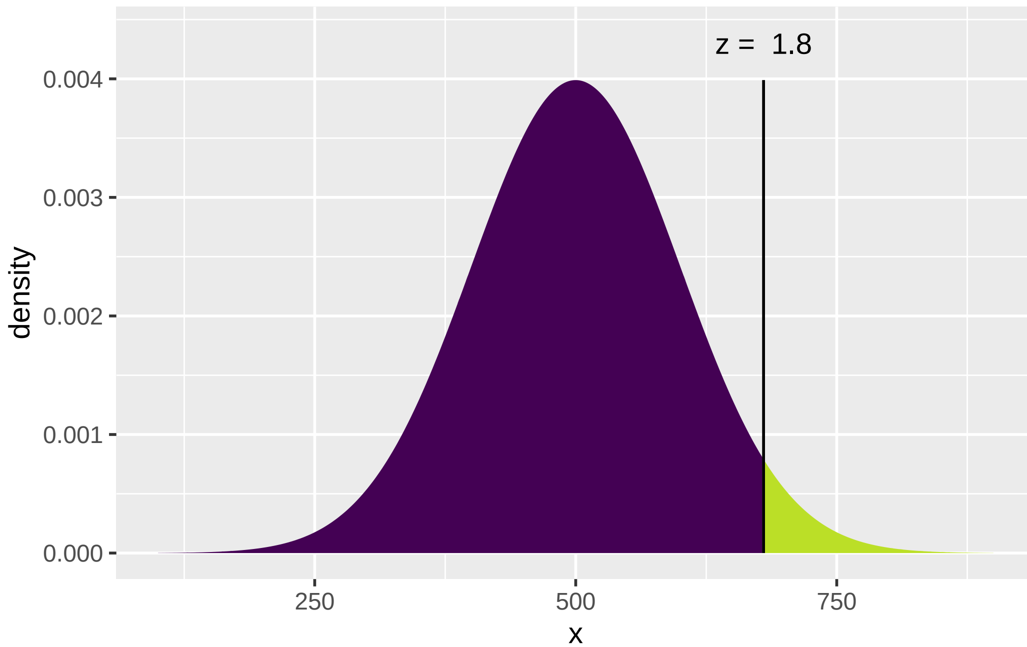
The **qnorm()** function finds the inverse of normal probabilities.

```
xqnorm(0.964, mean = 500, sd = 100) # inverse of pnorm()
```

```
If X ~ N(500, 100), then

    P(X <= 679.9118) = 0.964

    P(X >  679.9118) = 0.036
```




```
[1] 679.9118
```

```
qnorm(0.964, mean = 0, sd = 1) # what is the z-score?
```

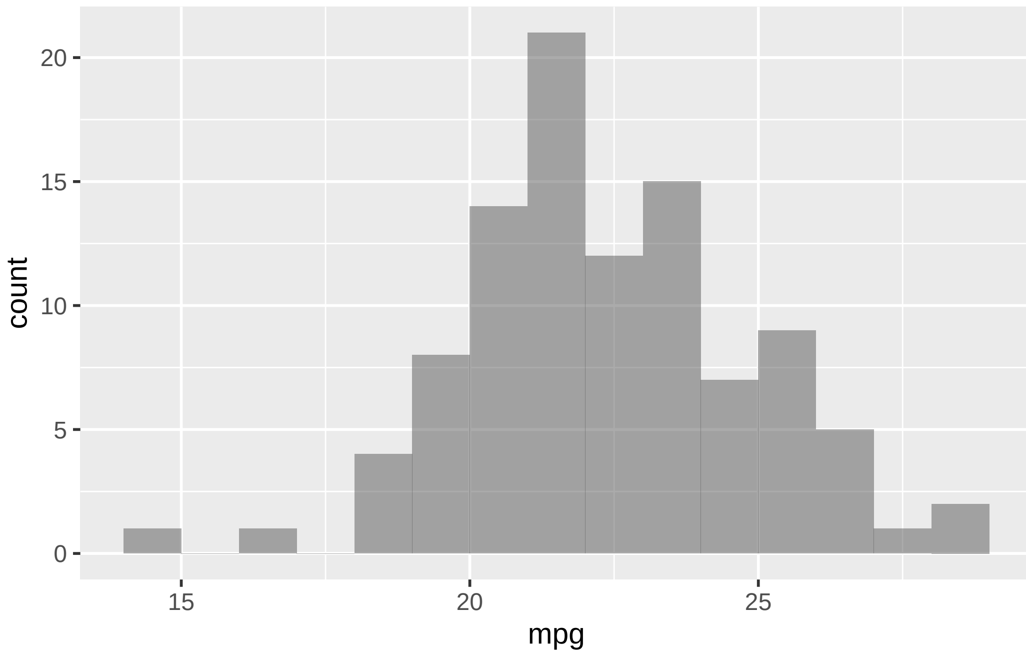
```
[1] 1.799118
```

See examples on pages 136-140.

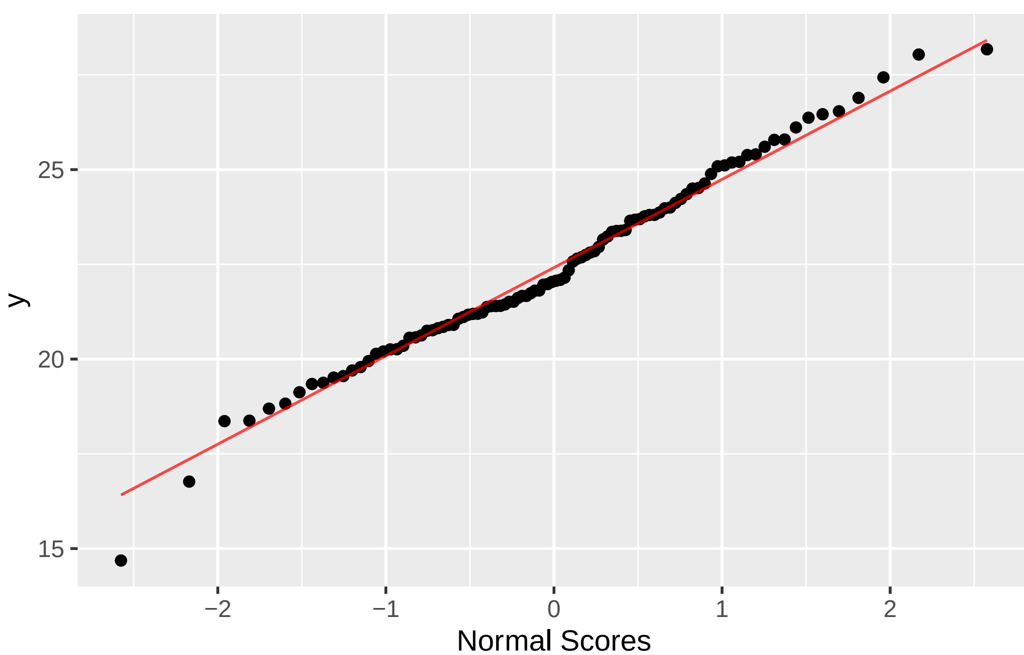
Section 5.5: Normal Probability Plots

We begin by reading in the data.

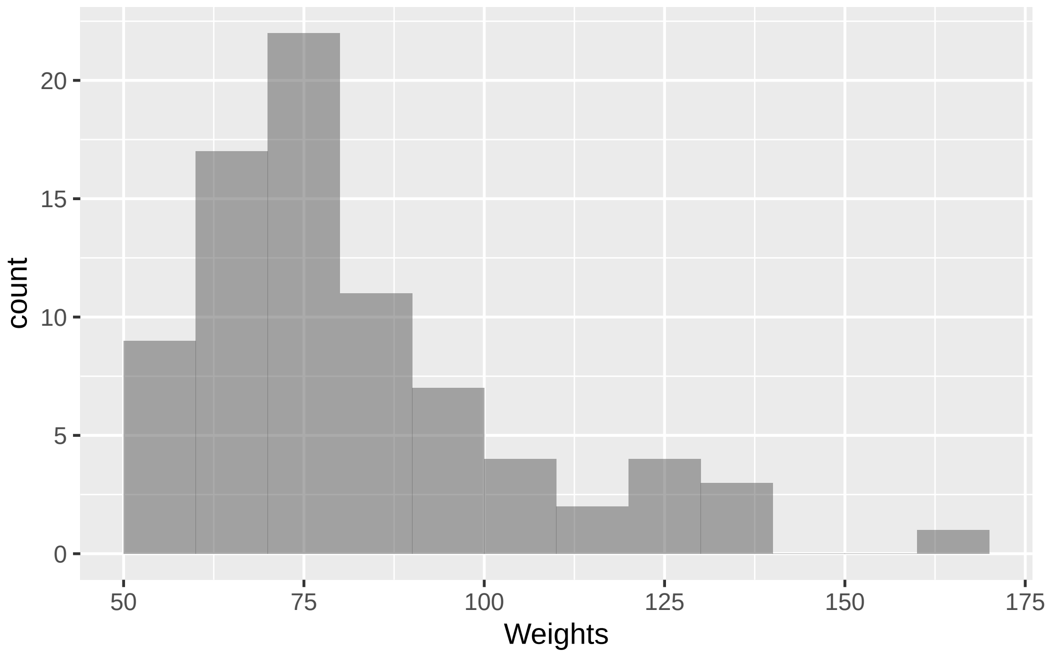
```
Nissan <- read_csv("http://nhorton.people.amherst.edu/is5/data/Nissan.csv")
# Figure 5.10, page 141
gf_histogram(~ mpg, data = Nissan, binwidth = 1, center = .5)
```



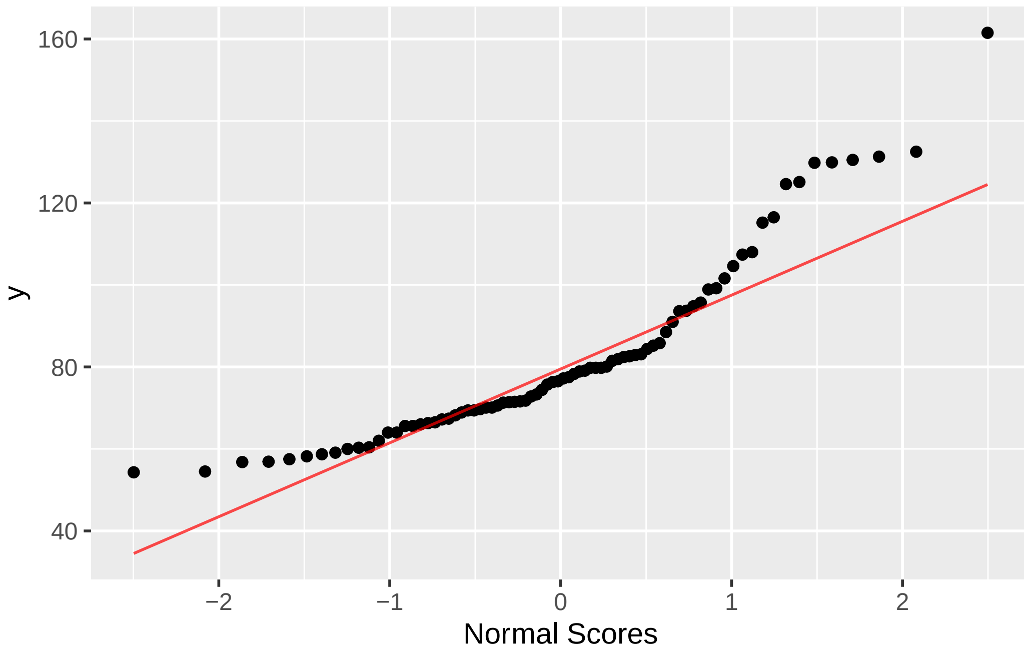

```
gf_qq(~ mpg, data = Nissan, xlab = "Normal Scores") |>
  gf_qqline(linetype = "solid", color = "red")
```



```
# Figure 5.11
gf_histogram(~ weight_in_kg, data = MenWeight, xlab = "Weights", binwidth = 10, center = 5)
```




```
gf_qq(~ weight_in_kg, data = MenWeight, xlab = "Normal Scores") |>
  gf_qqline(linetype = "solid", color = "red")
```


160
120
80
40
y
−2
−1
0
1
2
Normal Scores